Graph Grammars: True Concurrency Semantics and Refinement

Leila Ribeiro
Universidade Federal do Rio Grande do Sul
Outline

• Graph Grammars
• Semantics of graph grammars
• Refinements
• Future Work
Graph Grammars (GGs)

- Like Chomsky grammars, using GRAPHS instead of strings
- Data driven
- Visual language
- Few, but powerful concepts
- Rich theory, specially concerning concurrency
GGs: Applications

Biology: Describing plant and cell growth (L-systems)
GG: Applications

Specification of concurrent systems

states: graphs
actions: rules
GG: Applications

- Specification of model refactoring
- Efficient implementation of logic programming (using the graph structure to represent variable sharing)
- Pattern recognition/generation
- ...
GGs: Graph?

- Directed/undirected graphs
- Labeled graphs
- Typed graphs
- Attributed graphs
- Algebras
- More general kinds of graphs...
GGs: Graph?

- Directed/undirected graphs
- Labeled graphs

The algebraic approach to GGs is "relatively" independent on the kind of graph

- Algebras
- More general kinds of graphs...
Type graph

- Graph describing the types of vertices and edges allowed in states
Typed Graph

- Instances of the type graph

(graph morphism)
Rules

- LHS: pattern to be found in a state
- RHS: specified state change
- Relation LHS-RHS: specifies what does not change
Graph Grammar

GGs can be related by suitable (syntactical) morphisms that induce a behavior compatibility (simulation relation)
Rule Application

- Direct derivation or step

![Diagram showing the application of rule L1 and R1 through a pushout process](image)
Semantics

Sequential derivation

$\begin{align*}
L & \xrightarrow{r} R \\
\text{step 1} & \\
\text{INI} & \xrightarrow{} \text{OUT1=IN2}
\end{align*}$

$\begin{align*}
L' & \xrightarrow{r'} R' \\
\text{step 2} & \\
\text{OUT2=IN3} & \xrightarrow{} \text{OUT3=IN4}
\end{align*}$

$\begin{align*}
L'' & \xrightarrow{r''} R'' \\
\text{step 3} & \\
\text{OUT3=IN4} & \xrightarrow{\cdots \cdots}
\end{align*}$
Sequential derivation

\[ L \xrightarrow{r} R \]

\[ L' \xrightarrow{r'} R' \]

\[ L'' \xrightarrow{r''} R'' \]

step 1

step 2

step 3

\[ \text{INI} \xrightarrow{} OUT1=IN2 \xrightarrow{} OUT2=IN3 \xrightarrow{} OUT3=IN4 \xrightarrow{} \ldots \ldots \]
The semantics based on sequential derivations makes it difficult to reason about concurrency.
Concurrent Derivations

\[ L \xrightarrow{r} R \quad \text{step 1} \quad L' \xrightarrow{r'} R' \quad \text{step 2} \quad L'' \xrightarrow{r''} R'' \quad \text{step 3} \]

\[ \text{INIT} \rightarrow \text{OUT1} = \text{IN2} \ightarrow \text{OUT2} = \text{IN3} \rightarrow \text{OUT3} = \text{IN4} \]
Concurrent Derivations

\[ L \xrightarrow{r} R \]
\[ L' \xrightarrow{r'} R' \]
\[ L'' \xrightarrow{r''} R'' \]

INI → \( OUT1 = IN2 \) → \( OUT2 = IN3 \) → \( OUT3 = IN4 \)

step 1
step 2
step 3
Concurrent Derivations

\[ L \xrightarrow{r} R \]

\[ L' \xrightarrow{r'} R' \]

\[ L'' \xrightarrow{r''} R'' \]

\[ \text{INI} \rightarrow \text{OUT1} = \text{IN2} \rightarrow \text{OUT2} = \text{IN3} \rightarrow \text{OUT3} = \text{IN4} \]

Glue
Concurrent Derivations

Total order of steps substituted by a partial order of actions.
True Concurrency Semantics

• A concurrent derivation, also called deterministic process, is a special GG: there is a set of axioms to characterize such GGs, based on the overlappings of the rules in the type graph of the concurrent derivation.

• We can define non-deterministic processes by just dropping one of the axioms.

• The set of all non-deterministic processes of a GG ordered by inclusion has an upper bound, called unfolding, that is an object that describes all possible computations of this GG (but, contrastingly with sequential semantics, the same data element/action is never repeated). The unfolding can also be obtained operationally.

• This is analogous to corresponding construction for Petri nets, that are a special case of GGs.
True Concurrency Semantics

- There is a tight relationship between a GG and its Unfolding, given by an adjunction between the corresponding categories

\[
\text{SPO Graph} \xleftarrow{\perp} \text{Occurrence Grammars} \xrightarrow{\mathcal{U}_s} \text{Grammars}
\]
True Concurrency Semantics

• There is a tight relationship between a GG and its Unfolding, given by an adjunction between the corresponding categories

\[
\begin{align*}
\text{SPO Graph} & \xleftarrow{\bot} \text{Occurrence} \\
\text{Grammars} & \xrightarrow{\mathcal{U}_s} \text{Grammars}
\end{align*}
\]

• We could build a chain of adjunctions relating GGs to Event Structures and Domains, via unfoldings

\[
\begin{align*}
\text{SPO Graph} & \xleftarrow{\bot} \text{Occurrence} \xleftarrow{\mathcal{N}_s} \text{Asymmetric Event Structures} \xleftarrow{\mathcal{P}_a} \text{Domains} \\
\text{Grammars} & \xrightarrow{\mathcal{U}_s} \text{Grammars} \xrightarrow{\mathcal{E}_s} \text{Domains}
\end{align*}
\]

Which kind of refinement are we interested in?
Which kind of refinement are we interested in?

Abstract Behavior $\Rightarrow$ INTERFACE

Concrete implementation $\Rightarrow$ BODY

refinement

???

GTS
Refinement

Each production in the interface describes a complex interaction pattern implemented in the body.

We use abstract representation of computations, abstract representation of interaction patterns, transactions, and dependencies.
A *transactional graph transformation system* is a tuple \( GTS = (T, P, \pi, T_s) \), where

- \( T \) is a graph, called type graph;
- \( P \) is a set of production names;
- \( \pi \) is a function mapping production names to productions over \( T \) (a production is a pair of graph morphisms \( L \xleftarrow{l} K \xrightarrow{r} R \), where \( l \) is an inclusion and \( r \) is injective);
- \( T_s \) is a subgraph of \( T \), called the stable part of \( T \).
A transactional graph transformation system is a tuple $GTS = (T, P, \pi, T_s)$, where

- $T$ is a graph, called type graph;
- $P$ is a set of production names;
- $\pi$ is a function mapping production names to productions over $T$ (a production is a pair of graph morphisms $L \xleftarrow{l} K \xrightarrow{r} R$, where $l$ is an inclusion and $r$ is injective);
- $T_s$ is a subgraph of $T$, called the stable part of $T$.

A notion of observable part of a graph was added.
Example: TGTS

Type graph of a gas station specification
Example: TGTS

Rules of the gas station specification
Transactions

Given a tgt $\mathcal{G}$, a transaction is a derivation $\rho = G_0 \xrightarrow{q_1,m_1} G_1 \xrightarrow{q_2,m_2} \ldots \xrightarrow{q_n,m_n} G_n$ which satisfies the following properties:

1. $G_0$ and $G_n$ are stable graphs;

2. any intermediate graph $G_i$ ($i \neq 0, n$) is not stable;

3. the derivation $S(\rho)$ is equivalent to a direct derivation via a proper quotient of the production $q_1 + \ldots + q_n$ and a suitable match $m$, i.e., $S(G_0) \xrightarrow{PQ(q_1 + \ldots + q_n),m} S(G_n)$ is a derivation in $S(\mathcal{G})$.

4. the match $m$ is an isomorphism.
Transactions

Given a tmts $G$, a transaction is a derivation $\rho = G_0 \xrightarrow{q_1,m_1} G_1 \xrightarrow{q_2,m_2} \ldots \xrightarrow{q_n,m_n} G_n$
which satisfies the following properties:

1. $G_0$ and $G_n$ are stable graphs;
   - starts and ends in fully observable states
2. any intermediate graph $G_i$ ($i \neq 0, n$) is not stable;
   - cannot be divided into smaller transactions
3. the derivation $S(\rho)$ is equivalent to a direct derivation via a proper quotient of the production $q_1 + \ldots + q_n$ and a suitable match $m$, i.e., $S(G_0) \xrightarrow{PQ(q_1 + \ldots + q_n),m} S(G_n)$ is a derivation in $S(G)$.
   - dependencies between steps is based on non-observable items only
4. the match $m$ is an isomorphism.
   - the start graph contains exactly what is needed for the computation
Transactions

Given a tgs $\mathcal{G}$, a transaction is a derivation $\rho = G_0 \xrightarrow{q_1,m_1} G_1 \xrightarrow{q_2,m_2} \ldots \xrightarrow{q_n,m_n} G_n$ which satisfies the following properties:

1. $G_0$ and $G_n$ are stable graphs;
   - starts and ends in fully observable states
2. any intermediate graph $G_i$ $(i \neq 0, n)$ is not stable;
   - cannot be divided into smaller transactions
3. the match $m$ is unique and there is a derivation $S(G_0) \xrightarrow{PQ(q_1+\ldots+q_n),m} S(G_n)$ is a derivation in $S(\mathcal{G})$;
   - dependencies between steps is based on non-observable items only
4. the match $m$ is an isomorphism.
   - the start graph contains exactly what is needed for the computation
Example: Transaction
A transaction may “implement” a rule
A transaction may "implement" a rule

But a rule is too abstract to represent a complex interaction pattern
Idea: Add a dependency relation to rules such that

- The relation describes dependencies between elements of left- and right hand sides of a rule
- The dep-rule is "implementable" by a transaction
• Given a dep-rule, we can construct a set of rules that implement it (small step rules) such that the transaction generated by these rules is the original dep-rule.

• Given a dGTS, we can construct a “standard implementation”, that is a TGTS that implements all its rules.

\[ \text{dGTS} \rightarrow \text{TGTS} \]

Refinement or Implementation

Corradini, Foss, Ribeiro - WADT 2008
Further work

• How to prove (hopefully automatically) whether, given any TGTS, it is an implementation or refinement of a dGTS?

• How can we guarantee that a TGTS has no processes that do not correspond to (or are prefix of) transactions? No deadlock or divergence within TGTS...

• How to guarantee that an implementation does not add observable behavior (due, for example, to interference between interactions)